So far we have discussed the system as though it consisted of the computer and a memory. In fact, the computer itself consists of a set of circuits one of which is the memory. Thus the memory is more properly thought of as one of the parts of the computer. In this chapter, we will discuss one of the other portions, the CPU or Central Processing Unit. It is the CPU that actually does the "thinking" for the computer. As you might expect, it is also the most complex portion of the system.

The CPU itself can be broken into parts, and that is the way we will cover it here. Generally speaking these parts are additional memory locations that have some "extras" built into them. Because these particular memory circuits are not addressed or used in the same way as the locations we learned about in the last chapter, we will not refer to them as memory locations but registers. The CPU then is made up of a series of memory circuits called registers that perform certain functions within the computer. While this is a somewhat simplistic view it will serve us well for the purposes of this discussion.

The first of these registers is the Accumulator. In addition to being able to store two hex digits of information, it can also perform certain arithmetic operations on its contents. It is this register that performs the arithmetic and other mathematical functions of the computer. It gets the operands or numbers for these operations from the memory and likewise stores the results of the operations in the memory. Both the operations and the data transfers to and from memory take place under program control, that is, an instruction is required to load the accumulator with a number from memory, a second instruction is required to operate on the number, and a third is required to transfer the results or the answer back into the memory. Perhaps while the number is in the accumulator, we'll want to perform several operations on it instead of just one, in which case we'll have to have several more instructions. Both the instructions that tell the computer what to do with the number and the number itself are stored in the memory.

**A Program Example.** Let's try a very short little program just to get a feeling for how the computer operates. Please understand that this program is picked for its simplicity rather than its usefulness. We'll begin actually doing things in the next chapter. We will use the DCM mode to enter the program into the memory. Now however, it is time to get specific about instructions and how they work. We're going to very arbitrarily pick memory location 0100H as the address of the first instruction of the program. In that location we want to place the instruction that loads the accumulator with the data. Furthermore we will arbitrarily pick location 0200H as the place in memory that holds the data.

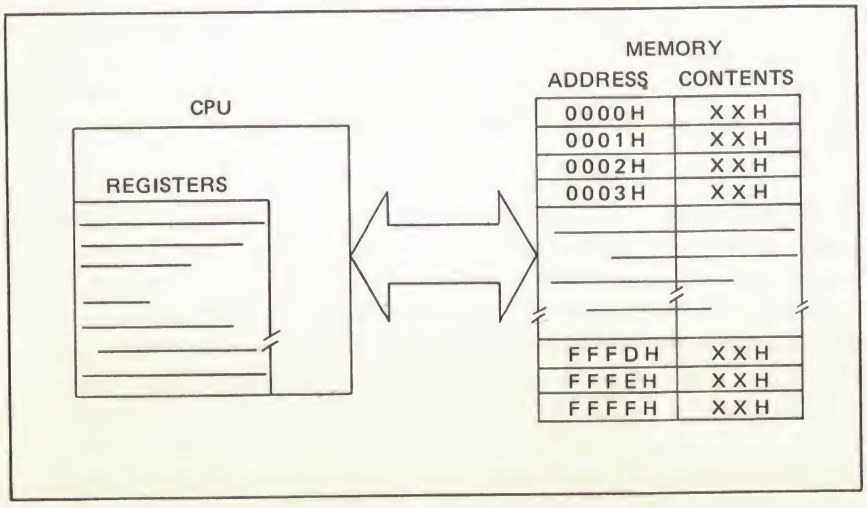



**Fig. 3-1.** The CPU stores both numbers and data in a set of numbered locations called the memory. Each location stores two-hex digit numbers. It takes a four-hex digit number called an address to specify any given location. The CPU itself has a set of memory circuits called registers.

The instruction that causes the accumulator to be loaded with data is the **Load Accumulator Direct** instruction, abbreviated LDA. Since computers don't understand the alphabet, let alone English, we'll have to look up the machine code equivalent of LDA, which happens to be 3A. This information is summarized conveniently on the Hex Card at the front of this book. At this point that card is probably about as easy to read as the London subway schedule, but believe us, you'll get to know it like the back of your hand. Rather than worry about the Hex Card right now, why not just take our word for the fact that the computer will be much happier with 3A than with LDA and let's go on with it.

How do we get 3A into location 0100H to start the program? You can probably guess, but if this isn't your morning try the DCM key.

| Enter: | See Displayed: |
|---|---|
| DCM 0 1 0 0 NEXT |  |
| 3 A |  |

The first instruction of our program is now safely locked away in memory. When the computer hits 0100H it will follow the instruction located there and load the accumulator with the data in memory. Wait a minute! Where in memory? We've got to tell the computer WHICH location contains the data to be loaded into the accumulator. We do that by using the next two memory locations after LDA, 0101H and 0102H, to store the data address. The computer automatically knows that when it finds the LDA instruction, or 3A in computerese, the next two memory locations will contain the address where the data is stored. Since we plan to keep the data at location 0200H we can load 02H and 00H into locations 0101H and 0102H to make the ia 7301 very happy. Just one more thing. When storing addresses in memory like this, the computer likes to

find them stored with the least significant two digits first, so we will place 00H in 0101H and 02H in 0102H. It's kind of like saying the President lives in the White House at 1600 Pennsylvania Avenue instead of Pennsylvania Ave 1600, LDA 00 02 instead of LDA 02 00.

| Enter: | See Displayed: |
|---|---|
| | 0100-3A |
| NXT | 0101-__ |
| 0 0 | 0101-00 |
| NXT | 0102-__ |
| 0 2 | 0102-02 |

Now that we have loaded the contents of location 0200H into the accumulator, let's do something with it! One obvious thing to do is increment it, that is, increase the value by one. If 07H was in the accumulator incrementing it should produce 08H; if 0FH was in the accumulator incrementing it would produce 10H, and so on. Fortunately there is an instruction available for this very purpose. It is **Increment Accumulator**, or INR A for short, and its hex code is 3CH. Load that instruction into location 0103H which is the next location that will be accessed as the computer executes the program.

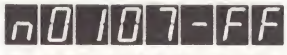
| Enter: | See Displayed: |
|---|---|
| | 0102-02 |
| NXT 3 C | 0103-3C |

If we could look into the accumulator at this point we would see that executing the INR A at 0103H had incremented the value in the accumulator. Since we can't do that we'll never know if the computer actually had followed these instructions. However we can load the new value in the accumulator back into the memory where we can examine it using the DCM mode. Any location will do but since we started with 0200H let's use it. The instruction that accomplishes this is called **Store Accumulator Direct** and its abreviation or mnemonic, as programmers call it, is STA. The hex code for STA is 32H but as in the case of LDA an additional two pairs of hex digits are required to specify exactly where in memory the contents of the accumulator are to be stored. Again, these are listed with the two least significant hex digits first, so that STA 0200H is actually loaded into the memory in the order STA, 00 and 02.

| Enter: | See Displayed: |
|---|---|
| |  |
| NXT 3 2 |  |
| NXT 0 0 |  |
| NXT 0 2 |  |

If we ended our program here everything would probably be all right. The only problem is that the computer would go on right past our last instruction in 0106H and try to execute all of the instructions from 0107H on. Of course you and I know that there's nothing in those locations except the random data that just happened to be there when the computer was turned on, but the computer does not know that. We just can't have the computer running through the memory trying to execute random instructions. For one thing it's not a very professional way of doing things, but more importantly, it might destroy the contents of location 0200H in the process and we'd never be able to see the result of our program.

There are several ways to stop the computer from going any further. Our favorite method is to plant an FFH in the next location. This will cause the computer to stop and go into another mode, but before we do this you should be warned that this method will work only with the ia7301 system. Other microcomputer systems will interpret this in a different way which we will describe later. The advantages to this method of stopping the system are that it only requires one memory location. This method also provides a very visible sign that the system has, indeed, been stopped.

| Enter: | See Displayed: |
|---|---|
| NXT F F |  |
| NXT |  |

We need that last NXT in order to complete the loading process of FFH into 0107H. Summarizing the program:

```
0100    3A      LDA     0200H   ;Load the accumulator with the contents
0101    00                      ;of memory location 0200H.
0102    02                      ;
0103    3C      INR     A       ;Increment the accumulator.
0104    32      STA     0200H   ;Store the contents of the accumulator
0105    00                      ;in location 0200H.
0106    02                      ;
0107    FF      RST     7       ;Stop the computer.
```

At this point it's always a good idea to go back and quickly review the memory location comparing them against the written program to be sure they're correct. The NXT key makes short work of this check and will frequently catch errors that could cost you a considerable amount of

time trying to debug a program that simply wasn't entered correctly to begin with. Incidently, when you begin to write your own programs don't fall into the habit of simply writing instructions and hex codes without the addresses for each instruction. This will undoubtedly prove very tempting, especially when you have a sudden inspiration at 2AM and you've just got to try it. So you write the starting address, the first instruction and then a string of instructions to fill out the program. But if you omit the addresses after the initial starting address there will be no way to check whether all of the instructions were really loaded into the memory. If, on the other hand, the addresses are all written next to the corresponding instruction that goes into that location, an error of ommission is immediately obvious because the addresses appearing on the displays will differ from those in the written program. The time saved in leaving addresses off programs is no where as great as the cost in time of trying to debug an incomplete program.

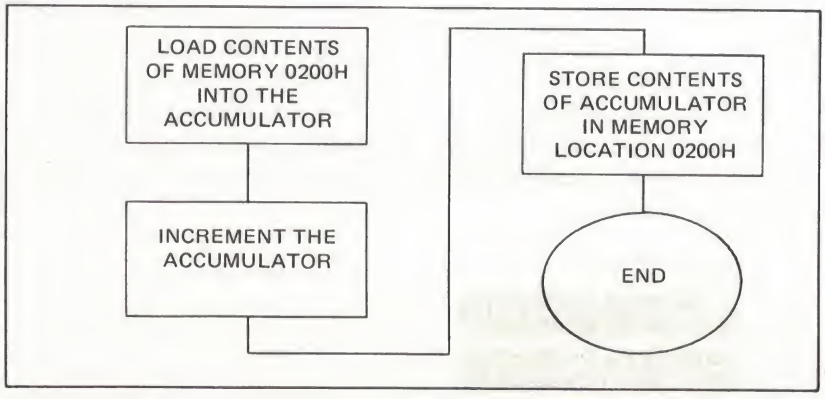
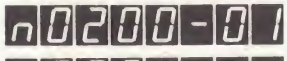



**Fig. 3-2.** The program can be summarized in a flow chart that defines the basic data flow as a function of time.

Of course before we go off and try to execute a program for incrementing some number in location 0200H, we should put an easily recognizable number there so we can see the change. Try 01H.

| Enter: | See Displayed: |
|---|---|
| DCM 0 2 0 0 NXT |  |
| 0 1 |  |
| NXT |  |

Now that the program is completely loaded into the computer, let's execute it. To do that we push CLR which resets an internal register to 0100H. This register, called the program counter, is what determines where the computer begins to execute the program. The monitor program has some instructions in it that cause the program counter to be automatically reset to 0100H every time CLR is pushed, so this is a convenient place to start writing the user's program. Later we will explore other ways to set the program counter to do other addresses so we can place our program anywhere in memory. Of course, pushing CLR will also cause the computer to reenter the DCM mode and display the DCM symbol and seven dashes. No matter, our program is in memory and the program counter has been set to location 0100H. All that remains is to push EXC and watch the computer execute the program.

| Enter: | See Displayed: |
|---|---|
| CLR |  |
| EXC |  |

Could you see a delay between the time you pressed EXC and the time that the displays popped up with the address 0108H on them? Probably not, and this should give you an idea of just how fast the computer actually is. Many instructions can be executed in a time so short that the human operator sees the results instantly with no measurable delay.

But did the program work? There is only one way to find out and that is to check location 0200H and see if the data has changed.

| Enter: | See Displayed: |
|---|---|
| DCM 0 2 0 0 NXT |  |

It worked! The data 01H that we had loaded into the memory has been incremented to 02H showing that the data was successfully fetched into the accumulator, incremented and replaced. We can execute the program again by just pressing CLR to reset the program counter and then EXC to run the program again.

| Enter: | See Displayed: |
|---|---|
| CLR EXC |  |
| DCM 0 2 0 0 NXT |  |

Once again, the contents of location 0200H have been incremented and then replaced in the memory.

**Single-Step Mode.** Placing an FFH in location 0107H caused the computer to stop when it reached this point. The displays come up with a new mode symbol, called the Single-Step Mode symbol, the address of the next instruction to be executed and what that instruction is. Since the last instruction was located at 0107H, the next will be at 0108H and it is this address that appears in the displays along with the contents of that location.

This brings up the subject of the Single-Step Mode which is one of the most useful features of the ia7301. The easiest way to understand it is to do it. Press CLR to reset the program counter and join us as we execute your program one step at a time.

| Enter: | See Displayed: |
|---|---|
| CLR | n - - - - - - - |

The program counter is now pointing at location 0100H which contains the LDA instruction that we loaded earlier. Executing that instruction will move the program counter up to location

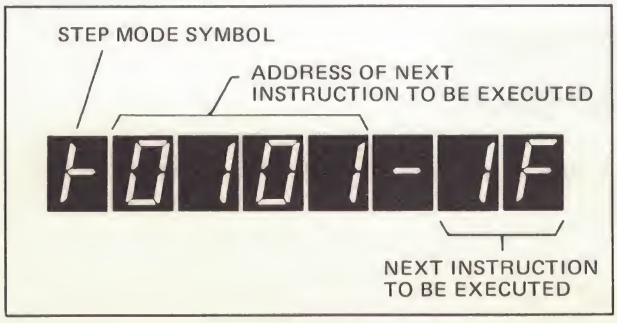



**Fig. 3-3.** The step mode displays both the address of the next instruction and the instruction itself.

0103H which contains the next instruction. This is because the LDA instruction also uses locations 0101H and 0102H to hold the memory address where the data will be found, a total of three memory locations. When the instruction has been completely executed the system will have gone through these three locations and the program counter will be pointing at the next location, 0103H. We can see this occur by pressing STEP which causes the computer to execute one instruction and stop, displaying the next instruction and its address.

| Enter: | See Displayed: |
|---|---|
| STEP |  |

Thus the computer executed the first instruction in the program which occupied locations 0100H, 0101H and 0102H and displayed the next instruction which appears at location 0103H. This instruction is 3CH which is the hex code for INR A. Since this instruction occupies only one location, pressing STEP will cause the program counter to be incremented by only one address so that it will now point to location 0104H.

| Enter: | See Displayed: |
|---|---|
| STEP |  |

32H is the hex code for STA which also requires three memory locations, one for the instruction and two for the address that is to receive the contents of the accumulator. Hence, pressing STEP again will increment the program counter by three so that it points to the last instruction, the FFH which we used to stop the computer.

| Enter: | See Displayed: |
|---|---|
| STEP |  |

Use of the STEP key can be a tremendous aid in debugging programs, but the real strength occurs when we combine it with the DCM and DCR keys. Let's try STEPping through the program again, this time using DCM and DCR to make some side trips along the way.

| Enter: | See Displayed: |
|---|---|
| CLR |  |
| DCM 0 2 0 0 NXT 0 8 |  |

So we know what's in location 0200H. Good. Now let's check the contents of the accumulator directly by pressing DCR .

| Enter: | See Displayed: |
|---|---|
| |  |
| NXT |  |
| DCR |  |

The DCR mode is used to display and change the contents of the various registers in the computer. The mode is accessed right from the keyboard via the DCR key, and as we can see, the first register displayed is the accumulator, denoted by the letter A. At this point in the execution of the program it contains 04H. Now press STEP to execute the LDA instruction.

| Enter: | See Displayed: |
|---|---|
| | uA----04 |
| STEP | H0103-3C |

The accumulator should now be loaded with the contents of the memory location 0200H which is 08H. We can recheck this by using DCR to examine the contents of the accumulator as it now stands after receiving the new data from the memory.

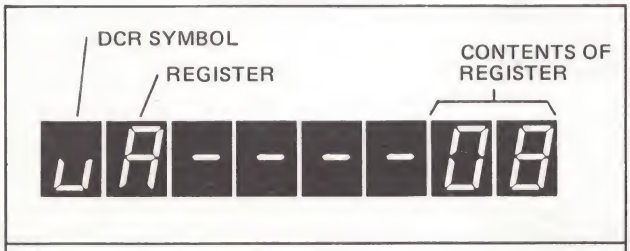



**Fig. 3-4.** The DCR mode uses two of the digits to display the contents of the register.

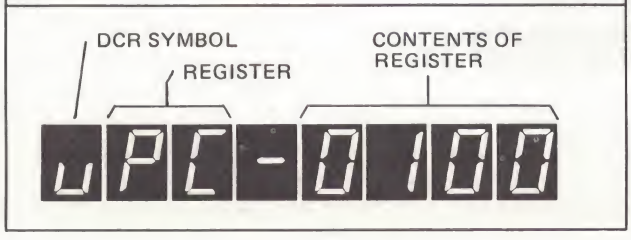
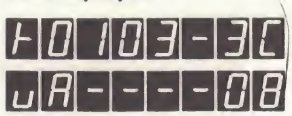



**Fig. 3-5.** The DCR mode also displays registers requiring four digits.

| Enter: | See Displayed: |
|---|---|
| |  |
| DCR | |

Obviously the accumulator has correctly accepted the data from the memory. Prior to pressing the DCR key the STEP display informed us that the next instruction to be executed is the INR A instruction in location 0103H. If we press STEP this instruction will be executed which will cause the accumulator to be incremented.

| Enter: | See Displayed: |
|---|---|
| |  |
| STEP | |

This is the next instruction which is the STA, but before executing that recheck the accumualtor.

| Enter: | See Displayed: |
|---|---|
| |  |
| DCR | |

The accumulator has been incremented as the program directed. There is one last instruction which we can execute by pressing STEP . This causes the STA to be implemented: we should be able to see the results by examining the contents of location 0200H. There we will find 09H showing that the incremented data has been successfully loaded back into the memory and the program is complete.

| Enter: | See Displayed: |
|---|---|
| STEP |  |
| DCM 0 2 0 0 NXT | |

**The Restart Instructions.** The CPU uses a system of interrupts to allow programs to be interrupted so that the computer can be used for higher level tasks. One of these interrupts comes into play whenever the STEP key is pressed. That key is electrically connected into the system so that it jams an FFH into the CPU instead of the instruction that would have normally been read in the execution of the program. FFH is a special instruction belonging to a class called Restart instructions. Altogether there are eight of these, although only one is used in the ia7301, RST 7 or FFH. Whenever this instruction is encountered by the CPU it automatically causes the program to be suspended while the computer begins to execute a special part of the monitor program that causes the Single-Step mode to be entered. While the Single-Step mode is normally entered via the artificial FFH that is jammed into the CPU, the mode can also be entered by placing an FFH into the memory so that it is encountered by the computer in the course of executing a program. When this happens, the jump to monitor occurs just as surely as though STEP had been pressed. We will discuss Restart instructions more in a later chapter.

**The Program Counter.** We have discussed the fact that the program counter must be set to 0100H if the computer is to execute a program that begins at that location. What we have not discussed is the fact that the computer uses the program counter during the course of executing each instruction in the program. It is the program counter that keeps track of which instruction is to be executed next. To do this the program counter must be incremented by one, two or

three addresses after each instruction is executed. That way it will be pointing to the first location of the next instruction when the computer comes back to it and asks for the location of the next instruction in the program.

The ia7301 makes provisions for examining the program counter during the course of executing a program in much the same way that the accumulator was examined. We have already seen how pressing DCR causes the contents of the accumulator to appear on the displays. Actually the accumulator is just the first of a series of registers that are displayed when the DCR key is pressed. Why didn't we see any of the others? Because the accumulator is always the first of the series to be displayed and before going on to any of the others we always pressed STEP and went on. The rest of the registers are accessible via the NXT key. Just as NXT causes the next memory location to be accessed when the computer is in the DCM mode, it causes the next working register in the CPU to be accessed when in the DCR mode. The computer treats the registers as though they are arranged on a carousel. Entry to the carousel always occurs through the DCR key and always at the same point, the accumulator. If we press NXT at this time, the carousel will rotate one position so that the next register, the flag register, appears on the displays. Each time NXT is pressed the carousel rotates one postion and a new register is displayed. Eventually the accumulator will be rotated around to the display position and will reappear once again.

This can be demonstrated using the same program that is already loaded.

| Enter: | See Displayed: |
|---|---|
| | 02000-09 |
| DCR | A----09 |
| NXT | F---- |

This is the flag register. It contains information such as whether or not the last operation produces a zero or whether the contents of the accumulator are positive or negative.

| Enter: | See Displayed: |
|---|---|
| | uF---- |
| NXT | ub---- |
| NXT | uC---- |
| NXT | ud---- |
| NXT | uE---- |
| NXT | uL---- |
| NXT | uH---- |

All of these registers are memory circuits within the CPU and will be used by the computer to perform certain specialized tasks. We can use them too, in much the same way that we used the accumulator in our program. All of the registers are tied to specific instructions that will utilize the registers in various ways. Much of our study of programming will be in learning the ways to make these registers work for us to accomplish our tasks.

All of the registers displayed so far have been capable of containing two hex digits. If we press NXT once more we revolve the carousel around to the four hex digit registers. The first of these is the stack pointer. As we shall see later proper use of the stack pointer allows the computer to go back and execute certain portions of the program over without having to use up memory to

duplicate the repeated portion. Notice that the stack pointer contains four hex digits. This is because the content of the stack pointer is an address and as you know, an address requires four digits.

| Enter: | See Displayed: |
|---|---|
| | uH----<br>uSP- |
| NXT | |

We'll come back to the stack pointer shortly. Right now, we are primarily interested in the next register which is the program counter. We have been discussing this register, but we have never actually seen it displayed before.

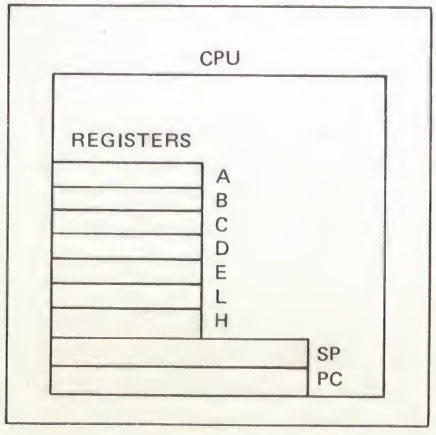
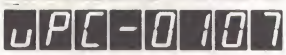
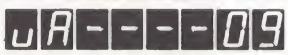
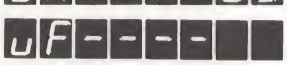


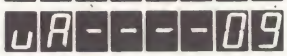
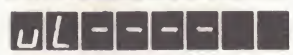
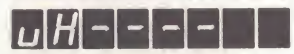
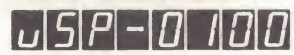
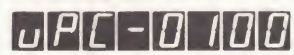

**Fig. 3-6.** Of the ten registers in the CPU, eight of them store two-hex digit numbers and two of them, the SP and the PC, store 4-hex digit numbers.

| Enter: | See Displayed: |
|---|---|
| |  |
| NXT |  |

The program counter at this point contains the address 0107H, which is where we left off in the execution of our program. This location contained FFH which was the RST instruction that caused the computer to stop and enter the single-step mode. The program counter is the last register on the carousel and if we press NXT once more the accumulator will be rotated into position to be displayed. We have come full circle and further depressions of the NXT key will only serve to rotate the carousel of registers around again.

| Enter: | See Displayed: |
|---|---|
| |  |
| NXT |  |
| NXT |  |

We have repeatedly made a point of the fact that pressing CLR causes the program counter to be reset to 0100H, but up until now you have merely taken our word for it. Now DCR allows us to see the resetting action of the CLR key directly. A second ago we saw that the program counter was set to 0107H.

| Enter: | See Displayed: |
|---|---|
| |  |
| CLR |  |
| DCR |  |
| NXT |  |
| NXT |  |
| NXT |  |
| NXT |  |
| NXT |  |
| NXT |  |
| NXT |  |
| NXT |  |
| NXT |  |

Here is proof positive of the action of the CLR key in setting the program counter to 0100H. Notice, by the way, that it also sets the stack pointer to 0100H. We will make use of this fact

later. For now, the fact that the program counter is set to 0100H whenever CLR is pressed makes 0100H a very convenient place to start our programs. If we start them at any other place than 0100H we will have to find some way to initialize the program counter to that starting address. By choosing 0100H as our starting address the setting of the program counter to this value is as easy as pressing CLR .

The computer operates by using the program counter as a pointer that keeps track of where the next instruction to be executed is. The fact that it now points to 0100H means the computer will find the next, and in this case the first, instruction at 0100H. If we press STEP it will cause an FFH to be jammed into the CPU which will in turn cause a jump to the Single-Step portion of the system monitor program. This program causes one instruction to be executed and the address of the next instruction to be executed and the address of the next instruction to be displayed along with the STEP mode symbol and the instruction itself. Since in this case the instruction requires three memory locations, one for LDA and two for the address containing the data to be loaded into the accumulator, the address displayed will be 0103H.

| Enter: | See Displayed: |
|---|---|
| |  |
| STEP | |

As before, pressing STEP caused the computer to execute the first instruction and display the next which appears at location 0103H. If we now use DCR to check the program counter we should find this fact reflected there.

| Enter: | See Displayed: |
|---|---|
| | H0103-3C |
| DCR | uA----09 |
| NXT | uF---- |
| NXT | ub---- |
| NXT | uC---- |
| NXT | ud---- |
| NXT | uE---- |
| NXT | uL---- |
| NXT | uH---- |
| NXT | uSP-0100 |
| NXT | uPC-0103 |

Here we see the computer working at the most fundamental level. The CPU uses the contents of the program counter to inform the memory which location it wishes accessed. It does this by sending the contents of the program counter out on a special set of wires called the address bus. The memory responds to that request by using the address on the address bus to locate one and only one set of circuits in the memory. The contents of this location are placed on a second set of wires called the data bus. The CPU picks this data off the data bus, interprets it as an instruc-

tion and executes it. The whole cycle is called the instruction fetch cycle; the CPU is said to have fetched an instruction from memory. The program counter is then incremented so that it will point to the next instruction.

Some instructions require more than one location; in fact, LDA is just such a case. It requires one location to hold the instruction itself plus two additional locations to hold the address of the data in memory. After retrieving the first two hex digits from the memory during the instruction fetch cycle that accesses location 0100H, the CPU decodes the machine code 3A contained there. In the process of decoding it, the CPU will discover that this particular instruction requires an address in order to be properly executed. Since addresses for instructions are always stored in memory immediately following the instruction itself, the CPU will perform two more fetches from memory. It does this by incrementing the program counter to 0101H and sending that out

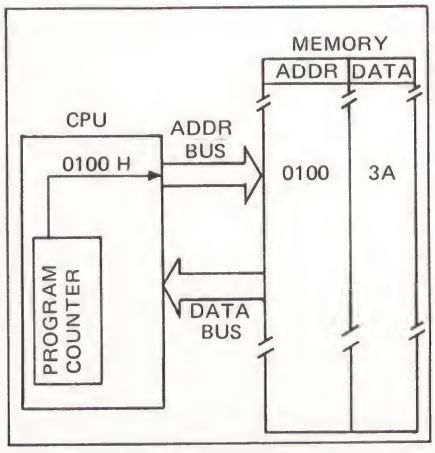



**Fig. 3-7.** During the first phase of the instruction fetch cycle, the contents of the program counter are sent to the memory over the address bus.

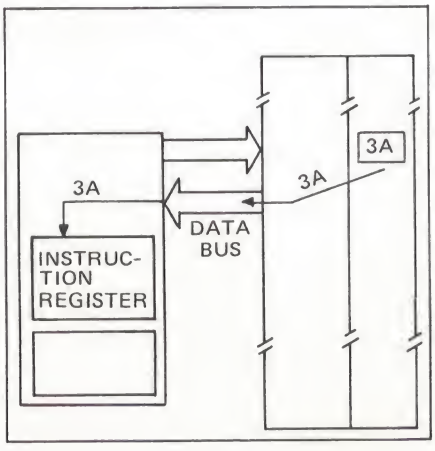



**Fig. 3-8.** During the second phase of the instruction fetch cycle, the contents of the addressed memory location are sent back to the CPU over the data bus. This does not affect the data in the addressed memory location which remains unchanged.

as an address to the memory. It stores the data that is returned by the memory in a special temporary holding register within the CPU. The program counter is then incremented again to 0102H and this address sent out on the address bus. The data retrieved during this second operation is placed within the temporary holding register next to the first data. The two batches of data, when taken together, make up a four hex-digit address which is then used by the CPU to execute the instruction at 0100H. Thus the program counter is used for fetching all instructions and data to the CPU; once the information enters the CPU it will be stored in one of two places depending upon whether the information is to be treated as an instruction or as data.

Pressing STEP at this time will cause the instruction at 0103H to be executed. Since this instruction only requires one memory location, the program counter will only be incremented by

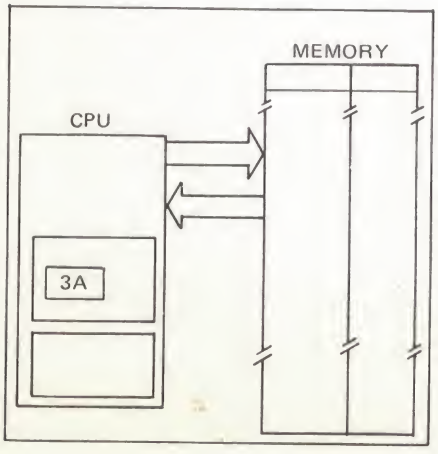
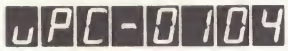
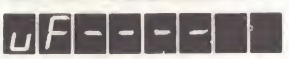
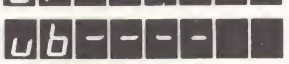
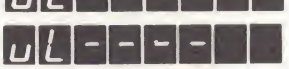
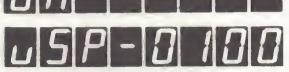
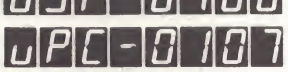



**Fig. 3-9.** During the third and final phase of the instruction fetch cycle, the instruction that was retrieved from memory is executed by the CPU. Some instructions are executed entirely within the confines of the CPU while others will use data stored in the memory.

one to 0104H. As before, we can check this by using DCR to examine the registers in the CPU.

| Enter: | See Displayed: |
|---|---|
| | uPC-0103 |
| STEP | H0104-32 |
| DCR | uA----0A |
| NXT | uF---- |
| NXT | ub---- |
| NXT | uC---- |
| NXT | ud---- |
| NXT | uE---- |
| NXT | uL---- |
| NXT | uH---- |
| NXT | uSP-0100 |
| NXT | uPC-0104 |

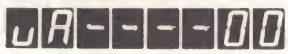
The next instruction is the STA 0200H which requires three memory locations. Pressing STEP should therefore increment the program counter by three so that it contains the instruction at location 0107H.

| Enter: | See Displayed: |
|---|---|
| |  |
| STEP |  |
| DCR |  |
| NXT |  |
| NXT |  |
| NXT |  |
| NXT |  |
| NXT |  |
| NXT |  |
| NXT |  |
| NXT |  |
| NXT |  |

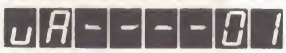
**Changing Registers On The Fly.** By now you should be thoroughly acquainted with the various registers in the CPU. There remains only one thing that should be discussed before going on to our first real program. The ia7301 has the ability to change the contents of any of the registers in the CPU while the system is in either the DCM, DCR or STEP modes without destroying the continuity of the program. The easiest way to understand this very useful function is to try it. Clear the computer to reset the program counter and check the contents of the accumulator using the DCR key. First, however we have to advance the program past the point that the contents of 0200H are loaded into the accumulator. Otherwise the program itself will erase any number that we load into the accumulator via the DCR key. To do this press STEP once so that the program counter is advanced to location 0103H.

| Enter: | See Displayed: |
|---|---|
| |  |
| CLR |  |
| STEP |  |
| DCR |  |

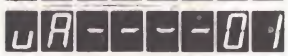
At this point we have always gone on to NXT or STEP , but if, instead, we press the hexadecimal keys on the keyboard, that data will be automatically loaded into the register. Let's use that fact to reset the accumulator to 00H before we increment it by our program. As in the DCM mode, the displayed information is not actually loaded into the register until NXT is pressed to cue the computer that the data on the displays is correct.

| Enter: | See Displayed: |
|---|---|
| |  |
| 0 0 |  |
| NXT |  |

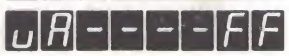
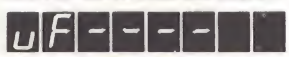
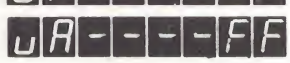
Does the accumulator really now contain 00H? The easiest way to find out is to execute the program and then check memory location 0200H. If our data entry into the accumulator through the DCR mode worked, the memory should now contain 01H since the program would have incremented the 00H in the accumulator and loaded this new value into 0200H.

| Enter: | See Displayed: |
|---|---|
| |  |
| EXC |  |
| DCR |  |
| DCM |  |
| 0 2 0 0 |  |
| NXT |  |

Was there ever any doubt? Using this technique we can load data into any of the registers that are displayed by the DCR mode. Don't forget to press NXT after entering the data or the computer won't really load it into the register. This is easy to illustrate.

| Enter | See Displayed |
|---|---|
| |  |
| DCR |  |
| F F |  |
| DCR |  |

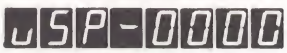
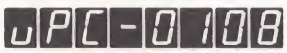
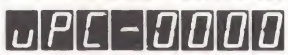
Thus, even though the data FFH appeared on the displays as though it were in the accumulator, this data was never really loaded into the register since NXT was not pressed, Pressing DCR the second time caused the DCR mode to be reentered displaying the accumulator. Had NXT been pressed prior to the second DCR depression, the data would have been correctly loaded.

| Enter: | See Displayed: |
|---|---|
| |  |
| F F |  |
| NXT |  |
| DCR |  |

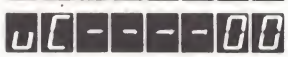
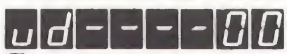
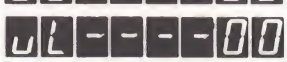
This time the data took!

This same procedure can be used to load any of the registers. Let's clear them all by loading 00H into each.

| Enter: | See Displayed: |
|---|---|
| | uA----FF |
| 0 0 | uA----00 |
| NXT | uF----__ |
| 0 0 | uF----00 |
| NXT | ub----__ |
| 0 0 | ub----00 |
| NXT | uC----__ |
| 0 0 | uC----00 |
| NXT | ud----__ |
| 0 0 | ud----00 |
| NXT | uE----__ |
| 0 0 | uE----00 |
| NXT | uL----__ |
| 0 0 | uL----00 |
| NXT | nH----__ |

| | |
|---|---|
| 0 0 |  |
| NXT |  |
| 0 0 0 0 |  |
| NXT |  |
| 0 0 0 0 |  |
| NXT |  |

Did the data take correctly? Just keep scanning through the registers and find out.

| Enter: | See Displayed: |
|---|---|
| |  |
| NXT |  |
| NXT |  |
| NXT |  |
| NXT |  |
| NXT |  |
| NXT |  |

NXT

NXT


SKILL III. AT THIS POINT YOU SHOULD FEEL COMFORTABLE WITH THE DCM, DCR, STEP AND EXECUTE KEYS AS WELL AS THE HEXADECIMAL KEYS. YOU SHOULD HAVE A FIRM GRASP OF THE ROLE OF THE PROGRAM COUNTER IN DIRECTING THE COMPUTER THROUGH THE EXECUTION OF THE PROGRAM. YOU SHOULD BE ABLE TO SINGLE-STEP THROUGH A PROGRAM, AND AT THE SAME TIME BE ABLE TO CHECK THE REGISTERS AND MODIFY THEM BEFORE THE NEXT PROGRAM STEP.